第3章

2軸加速度センサの値をパソコンに取り込む装置の製作

PWM出力で外付け回路の少ないADXL213を利用

西村芳一

加速度センサの出力には，大きく分けて二つある．一つはアナログ信号を，もう一つはディジタル信号を出力する品種だ．アナログ信号を出力する品種は，外部でフィルタや増幅器などを持たせる必要がある．ここでは，加速度に応じてパルスの幅が増減するPWM（pulse width modulation）出力を持つ加速度センサを例に，パソコンにその値を取り込むまでを紹介する．

（編集部）

シリコン上に組み上げられた「マイクロメカ」を使ったMEMS（micro electro mechanical systems）は，本誌でもたびたび取り上げられ，以前からとても興味がある技術でした．しかし，実際の製品への応用となると，なかなか思いつかないものです．今回紹介する加速度センサは，カメラの手ぶれ防止や自動車など，意外と身近な商品に，知らないうちに応用が進んでいるようです．

なんとなく扱いにくい印象のあったデバイスですが，米国Analog Devices社の*X*-*Y*軸加速度センサ「ADXL213」（**写真1**）を使ってみる機会を得たので，使用例を紹介します．思いのほか簡単に扱えるので，これを機にいろいろな分野で利用してみたらいかがでしょうか．

写真1　ADXL213の評価ボードADXL213EB

1．パルス幅変調出力を持つADXL213のあらまし

図1にADXL213の内部ブロックを示します．直交する2軸方向の加速度を測定できます．振動などの動的加速度だけではなく，静的な重力加速度なども測定できます．実際，このセンサの測定できる範囲は±1.2gですから，重力加速度程度を測るのに向いています．実際に試したことはないのですが，地下の岩盤の組成の違いによって，地上の重力加速度は異なることが知られているので，重力計としてそれが検知できれば，個人的には面白いと思いました．

写真1のように約5mm角のセラミックLCC（leadless chip carrier）パッケージに収められています．小さな製品にも組み込める大きさです．

出力の分解能と周波数レスポンスはトレードオフの関係にあり，**図1**に示すC_xとC_yの値を変えることによって調整が可能です．コンデンサの値を大きくすると，分解能は上がりますが，レスポンスが悪くなります．ADXL213は使用目的に合わせて簡単に特性が変えられますから，とても便利です．

KeyWord　パルス幅変調，PWM，ADXL213EB，EP1C3，ByteBlaster II，RS-232-C，ハイパーターミナル，VHDL，MEMS

図1[1]　**2軸加速度センサ［ADXL213］の内部ブロック**
PWM出力でマイコンなどとの接続が容易．

図2[2]　**加速度センサ評価ボードADXL213EBの回路図**

センサからの出力は，マイコンなどと簡単に接続できるように，PWM（pulse width modulation）の出力になっています．特にA-Dコンバータがなくても接続できるので，とても使いやすくなっています．

図1に示すPWMのパルスにおいて，T_2の周期はR_{set}で決められて常に一定です．ただし外部の抵抗を変えることにより，周期を変えることが可能です．決められた速度で動作するマイコンにつなぐ場合には，そのマイコンのクロック速度に合わせることができ，とても便利な仕様となっています．T_1の幅は，加速度の大きさで変化します．従ってT_1とT_2の比を計算すれば，最終的な加速度が分かります．

2．ADXL213の出力をパソコンに表示するための構成

これから，実際にこのデバイスを使って動作実験を行ってみることにします．入門者でも簡単に再現できるように，できるだけ簡単な回路を使うことにしました．また，新たに基板を作らなくても実験できるように，出来合いの基板を組み合わせて使うことにします．

● 実装を考慮し評価ボード「ADXL213EB」を入手した

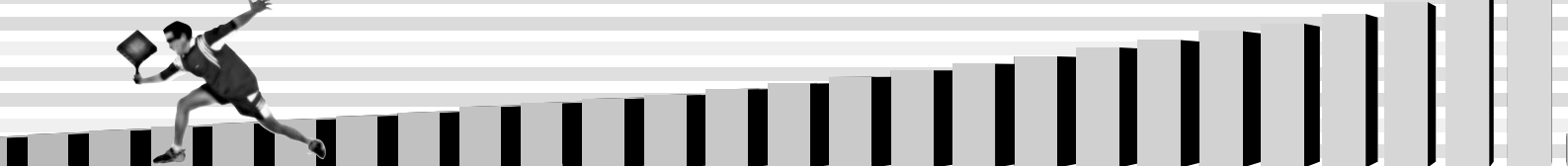
写真1から分かるように，ADXL213のパッケージはとても小さいので，ユニバーサル基板に手はんだで配線するのは苦労しそうです．そこで普通なら基板を作らないといけなそうですが，幸いにもAnalog Devices社から**写真1**のような評価基板が売られています[編集部注]．**図2**のような本当に基本的な回路ですが，バラック配線をすることを考えれば，とても便利です．そこで本基板を購入しました．

基板上にはR_{set}，C_2，C_3を取り付けるランドがあります．まず出力のPWMの基本周期を決めるためのR_{set}を取り付けます．周期T_2［s］は，

$$T_2 = R_{set}/125 \times 10^6 \,[\Omega]$$

の関係式で決まります．ここでは約1kHzで設計するので，R_{set}を120kΩとしました．C_2とC_3はX軸とY軸それぞれの周波数応答特性を決めます．それらの関係式を**図3**に示しました．ここでは50Hz程度の帯域幅を得るため，どこでも簡単に入手可能な0.1μFの積層セラミック・コンデンサを取り付けました．

ADXL213の動作電圧範囲は広く，仕様書によれば3～

編集部注：米国Analog Devices社の製品に関する問い合わせは，下記の販売代理店までお願いします．
エー・ディ・エム（株）　http://www.adm.co.jp
グローバル電子（株）　http://www.gec-tokyo.co.jp
富士エレクトロニクス（株）　http://www.fujiele.co.jp
（株）マクニカ　テクスターカンパニー　http://www.tecstar.macnica.co.jp
なお，Analog Devices社のWebページから購入できる品種もあります．http://www.analog.com/jp/index.html

6Vの範囲で動作します．ここでは，その他の回路の動作電圧に合わせ，3.3Vを使っています．このような広い動作電圧範囲は，さまざまなICとインターフェースする際にとても便利です．

● FPGAは本誌2003年10月号付属基板を利用した

このADXL213の出力を取り込むデバイスとしては，普通はマイコンを考えるでしょう．マイコンのタイマを使い，そのパルス幅を測定することがすぐに考えられます．しかし，マイコンの場合は，簡単なプログラムだといっても，ソフトウェアを作りデバッグする行為が，筆者にとって面倒に思えました．そこで，FPGAによる実装を行うことにしました．デバッグもソフトウェアとは違い，シュミレーションで簡単に短時間で確認できます．実際，全体の設計およびデバッグは1日で終わりました．

次にFPGAを選ばなければなりません．幸いにも本誌には何回かFPGA基板が付属しています．その中から2003年10月号の付属基板を使うことにしました．本基板には米国Altera社の「EP1C3」というCyclonデバイスが搭載されています．今回の実験には規模が大き過ぎる感じもしましたが，一番手ごろなため，これを利用することにしました．本誌2007年7月号にも米国Xilinx社のFPGAが搭載された基板が付属しており，もちろんそれでも問題なく使えると思います．それらの基板とのインターフェースは自分で考える必要がありますが，簡単に置き換え可能です．

2003年10月号付属基板には電源用の3端子レギュレータも搭載され，とても使いやすくなっています．問題はクロックが付いていません．そこで，手元にあった20MHzの水晶発振器を**写真2**のように取り付けました．この20MHzの発振器は悪いことに5Vで動作します．FPGAとADXL213は3.3Vで動作させるので，5Vが別に必要になります．そこで，電源供給を5Vにし，そこから3.3Vのレギュレータにつなぐことにしました．

CyclonはSRAMタイプのFPGAなので，電源投入時に回路データのダウンロードが必要です．JTAGを使ってダウンロードすることもできますが，出来上がってからいちいちパソコンとつないでデータをダウンロードするのは面倒なので，ダウンロード用のシリアルROMを取り付けることにしました．幸い，ROM用のパターンはあらかじめ引いてありますから，部品をはんだ付けするだけです．手持ちの「EPCS4」を取り付けました．

ROMへのプログラムはダウンロード用のケーブル（ByteBlaster II）を使います．ROMを基板に取り付けたまま何度でも書き込めます．そこでROM書き込み用に10ピンのヘッダを取り付けました．

f_{-3db}＝5［μF］/$C_{(X,\ Y)}$ ← C_2またはC_3の値

帯域幅［Hz］	コンデンサ容量［μF］
1	4.7
10	0.47
50	0.10
100	0.05
200	0.027
500	0.01

図3[1]　センサの周波数特性を設定するためのめやす
外付けコンデンサの定数でX軸とY軸それぞれの帯域幅が決まる．

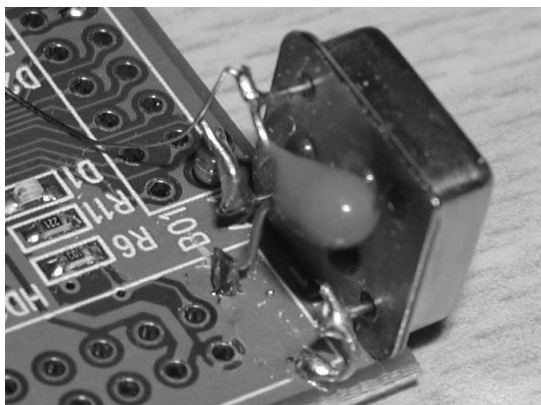



写真2　2003年10月号の付属基板に20MHzの水晶発振器を取り付けたようす

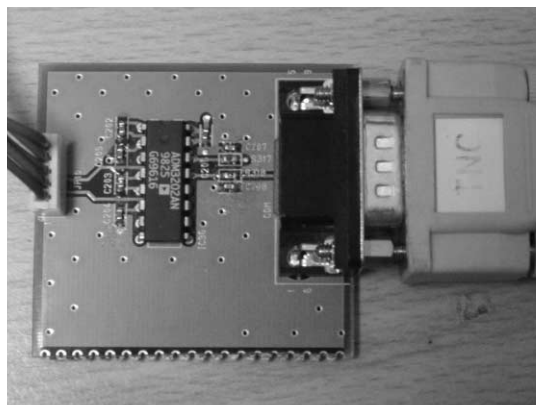



写真3　パソコンのシリアル通信ポートと2003年10月号付属基板を接続するレベル変換基板

● パソコンとはRS-232-Cで接続

FPGAの中で検出したデータを，何らかの手段で見える形にしなければなりません．そこで，パソコンとのインターフェースで最も簡単な，非同期シリアル通信を行うことにしました．そのため，FPGAの3.3Vの信号をRS-232-Cの信号レベルに変換する必要があります．

これまで，同じようなシリアル通信による回路デバッグを何度も行ってきたので，幸いにも手元に**写真3**のようなレベル変換基板があります．インターフェースは**図4**のような回路で，コネクタを介してTXDとRXD，3.3V電源，グラウンドをつなぐだけです．

変換基板には9ピンのDサブ・コネクタがついており，パソコンのCOM端子とつなげば，すぐに通信できるようになっています．ハンドシェイク・ラインはすべて，回路図のようにヌル接続となっています．

このような変換基板を一度作っておけば，これからの回路設計でも，デバッグ用インターフェースとして統一することにより，便利に使うことができます．レガシ・インターフェースといわれ，パソコンからはなくなる方向のインターフェースですが，設計現場においては，シリアル通信を使ったデバッグはまだまだ使われ続けるでしょう．

RS-232-Cと3.3VのインターフェースICは，Analog Devices社の「ADM3202A」を使いました．このICは広く使われており，東京・秋葉原の部品販売店（秋月通商など）でも安く簡単に入手することが可能です．

これまで紹介した回路をつなぎ合わせた回路を**図5**に示します．筆者の場合はシリアル・レベル変換基板が手元にあったので，その部分は配線の必要がなく手間が省けました．残りの配線は，慣れた人なら1時間もかからないくらいの簡単なものだと思います．

3．製作する装置の仕様を決める

これまで説明した回路を使い，具体的にFPGAの回路設計をはじめます．その前にどのような仕様でパソコンにセンサ出力を取り込むのか検討します．

基本的にはパソコンからのコマンドを解釈し，それに従って測定したADXL213のデータを取り出して加工し，パソコンに送り返す構造です．パソコンはハイパーターミナルなどといった汎用ソフトウェアを使い，シリアル通信を行います．FPGA内の処理は大きく分けて四つのブロックから構成されています．

● ADXL213からのデータの取り込み

ADXL213からはパルス幅変調されたデータが送られてきます．基本的にはそのパルス幅を20MHzのクロックで測定します．この回路定数では波形を1kHzの周期で繰り返す設定にしています．20［MHz］/1［kHz］＝20000となり，最大1/20000の分解能で測定できますが，ここでは10ビット（1024）の分解能で測定することにします．測定は1kHzの周期で，X軸とY軸を独立にそれぞれ測定します．データ出力は，加速度ゼロを中心に2の補数でプラスとマイナスの値です．

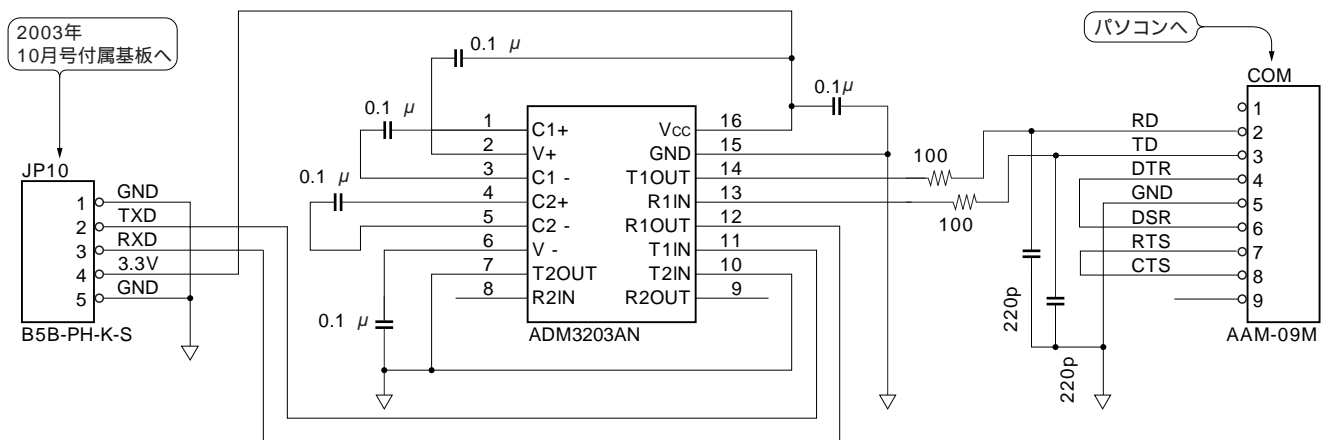



図4　パソコンのシリアル通信ポートと2003年10月号付属基板を接続するレベル変換基板の回路図
Cyclonの3.3Vの信号をRS-232-Cの信号レベルに変換する．

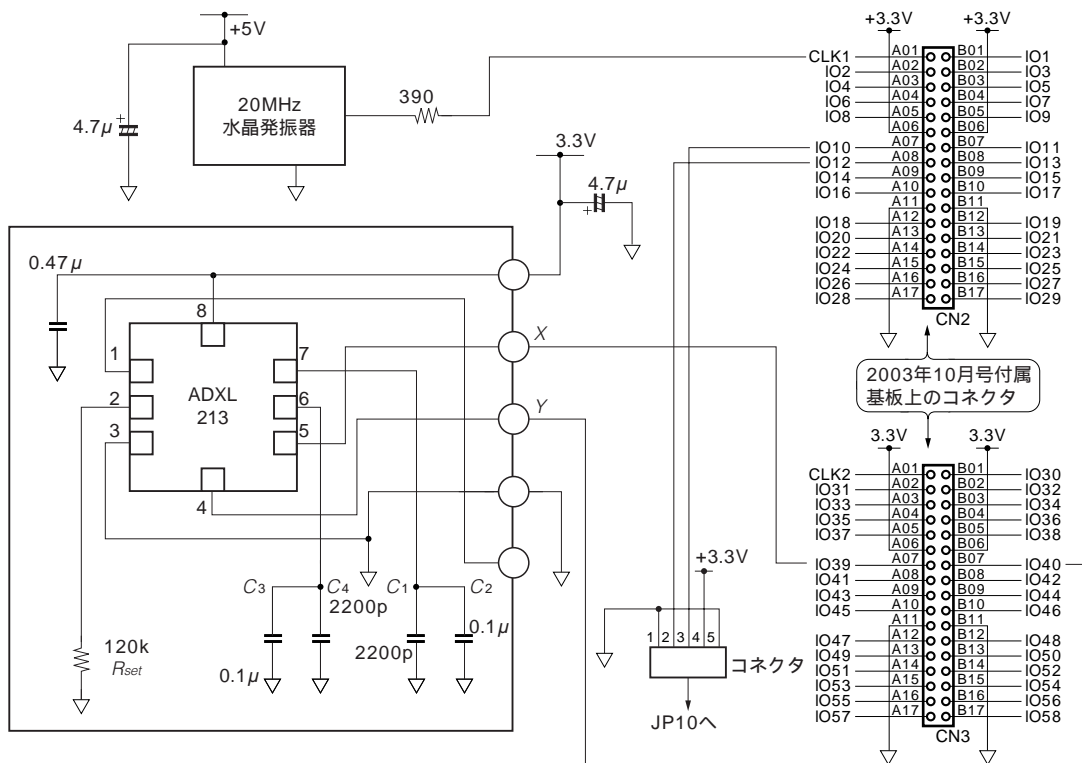
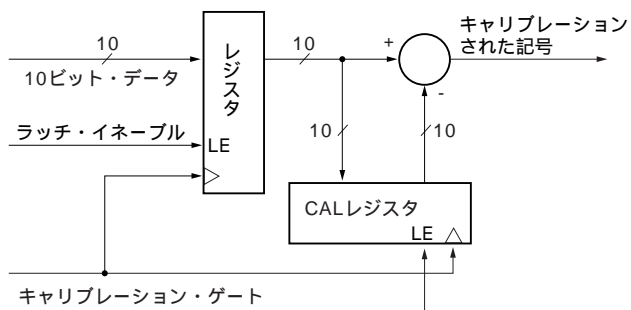



図5
加速度センサの出力をパソコンに表示する装置の回路図
2003年10月号付属基板の回路は割愛した．

● 加速度センサから取り込んだデータをどのように処理してパソコンへ送り出すのか

読み込んだだけのデータをそのまま出すのは味気ないですから，コマンドに従っていくつかの算術計算処理を行います．

● キャリブレーション

ICのばらつき，あるいはパルス幅のばらつきなどにより，加速度ゼロの絶対値をADXL213のパルス幅から得るのは難しいと思われます．そこで**図6**のようなゼロ・キャリブレーション回路を設けました．パソコンからのキャリブレーション・コマンドを受け取ると，その時点のADXL213の値をゼロとみなすように，出力にオフセットを持たせます．

例えばADXL213を静止した水平な台の上に置いたときは，重力加速度などが加わらないですから，ここを加速度ゼロと想定できます．そのときパソコンからキャリブレーション・コマンドを送ると，そこをゼロに設定します．

● キャプチャ

パソコンからコマンドを受け取った時点での，*X*軸と*Y*軸の値を内部で保持します．それ以降はADXL213からのデータを受け付けません．パソコンから読み出すときは，現在値にその値が読み出されます．

● 最大値保持

パソコンからコマンドを受け取ると，それ以降，加速度のプラス方向の最大値を内部に自動的に保持します．パソコンからの読み出しに対しては，それ以降，現在値ではなく，その最大値が読み出されます．

● 初期化

初期化のコマンドをパソコンから受け取ると，内部のモードを現在値読み取り状態に戻します．また，キャリブレーションのオフセットをゼロにクリアして，生のデータ

図6　ゼロ・キャリブレーション回路
パソコンからのキャリブレーション・コマンドを受け取ると，その時点のADXL213の値をゼロとみなすように，出力にオフセットを持たせる．

を読み出します．

● パソコンからのコマンドの種類

パソコンからの非同期シリアル・コマンドを受信し，それを解読し，内部の処理系のトリガをかけます．シリアル伝送速度は19.2kbpsです．シリアルのパラメータは，「8ビット，パリティなし，1ストップ・ビット」です．

元のクロックは20MHzなので，ぴったり19.2kbpsになりません．20［MHz］÷65÷16＝19.231［kbps］となりますが，十分吸収できる誤差の範囲でしょう．

パソコンからのコマンドには，**表1**に示す6種類があります．簡単な回路とするため，ASCII文字で1文字のコマンドとなっています．大文字または小文字，いずれでも受け付けます．つまり，パソコンの対応する文字キーを押したことを検出し，処理を行います．

● パソコンとのシリアル通信の設定

パソコンからのコマンドがデータを要求した場合，データ処理部で計算した値をASCII数字に変換し，19.2kbpsの速度で，シリアル信号で送り返します．シリアル通信のパラメータは受信部と同じで，「8ビット，パリティなし，1ストップ・ビット」です．

全体の処理の流れを**図7**に示します．データ処理部から受け取った2の補数の数値は，符号ビットが取り出された後，絶対値に変換されます．符号ビットを取り除いた後の絶対値は9ビット幅のデータです．従って0～511の範囲の数値になります．

このバイナリの数値を，3けたの10進数に変換します．変換した後，プラスとマイナスの符号と3けたの10進数の合計，四つのASCIIキャラクタをパソコン側に送ります．

表1　パソコンから受け取るコマンド

コマンド・キャラクタ	Hex	コマンド
“X” “x”	0x58 0x78	X軸のデータを読み出す
“Y” “y”	0x59 0x79	Y軸のデータを読み出す
“C” “c”	0x43 0x63	現在値をゼロにするようにキャリブレーションする
“M” “m”	0x4D 0x6D	最大値保持モードへ．正の加速度の最大値を保持
“S” “s”	0x53 0x73	現在値保持モードへ．現在値を保持
“R” “r”	0x52 0x72	現在値読み出しモードへ．キャリブレーションOFFセットをリセットする

パソコンのターミナルなどを利用すれば，ADXL213が検出した値を読み取ることができます．

4．VHDLによる設計

前項で説明した四つのブロックを，VHDLを使って回路記述します．FPGA開発ツールはAltera社のQuartus II Web Editionを使いました．メインのVHDL記述を**リスト1**に示します．ルートの記述では，それぞれのブロックに対応する，

① **Rex210**：ADXL213のパルス幅読み取り
② **Caldata**：データの処理部
③ **Rcsq**：シリアル・データ受信ならびにコマンド解釈
④ **Tcsq**：ASCII数値変換ならびにシリアル・データ送信

の四つの回路を単につなぎ合わせただけです．すべての回路は20MHzクロックの同期回路となっています．

シュミレーションのために，すべての回路に非同期リセットを設けています．実装するときも，FPGAのピンに

3

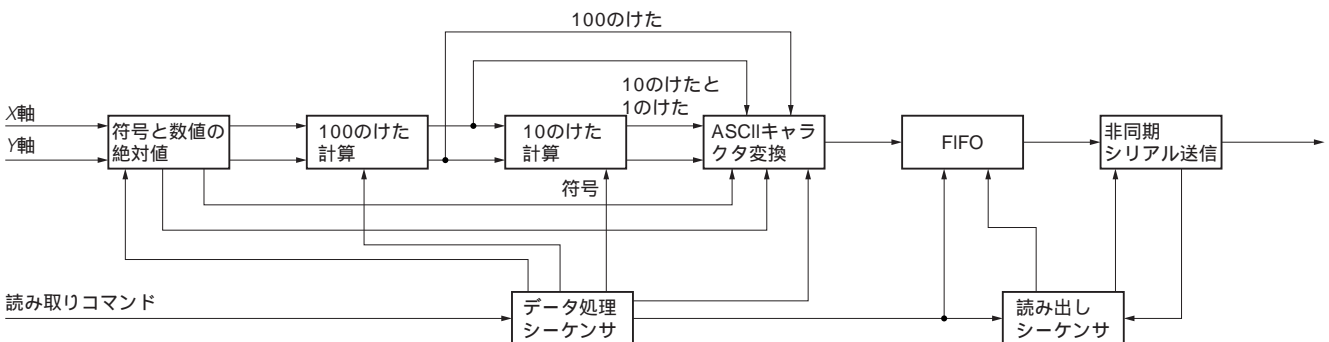



図7　ASCII数値変換およびシリアル・データ送信のブロック図
パソコンからのコマンドがデータを要求した場合，データ処理部で計算した値をASCII数字に変換し，19.2kbpsの速度で，シリアル信号で送り返す．

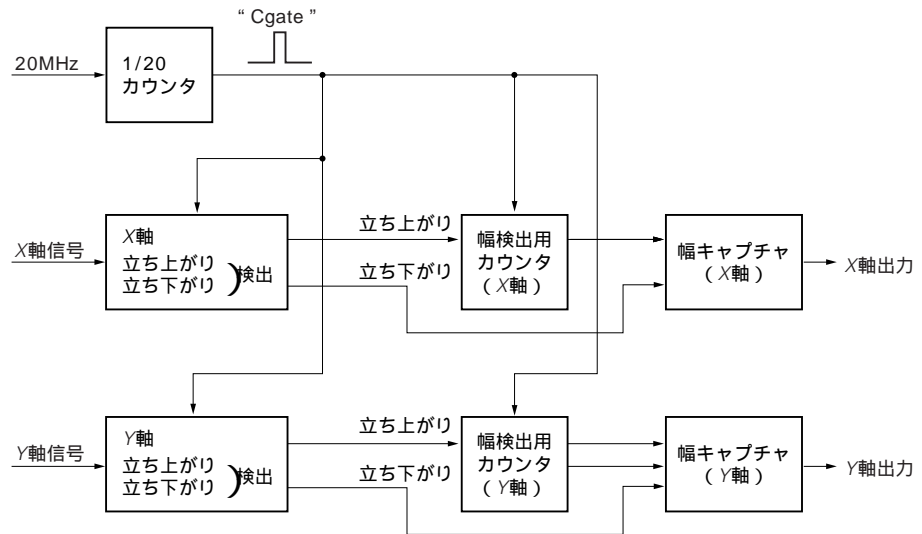
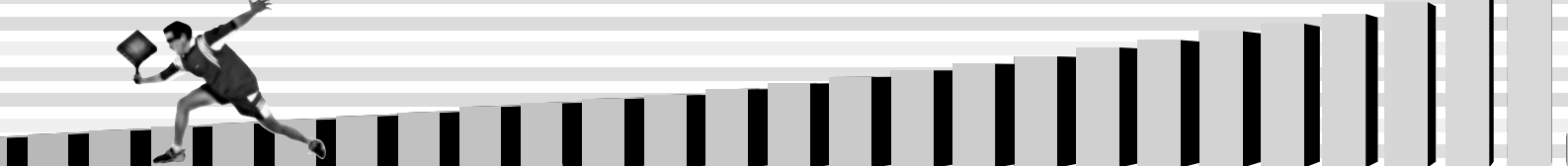



図8
パルス幅検出回路のブロック図
すべての回路は1MHzのゲート信号に同期して動作する．

リスト1　データ取得回路のVHDL記述

```
--------------------------------------------------------
--      AD213 process main
--              (C)2007,04,08 by Yoshikazu Nishimura
--------------------------------------------------------

LIBRARY ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity Ad213main is

  port
    (
      Mclk     : in  std_logic;  -- Master clock 20MHz
      Reset    : in  std_logic;  -- Master Reset
      Xin      : in  std_logic;  -- X axis data
      Yin      : in  std_logic;  -- Y axis data
      Rxd      : in  std_logic;  -- Serial input
      Txd      : out std_logic   -- Serial data
    );

end Ad213main;

ARCHITECTURE one OF Ad213main IS
  signal  Xdata,Ydata         : std_logic_vector(9
                                              downto 0);
  signal  Xout,Yout           : std_logic_vector(9
                                              downto 0);
  signal  Mode                : std_logic_vector(1
                                              downto 0);
  signal  Xsamp,Ysamp,Tgate   : std_logic;
  signal  Cal,Mtrig           : std_logic;
  signal  Xtrig,Ytrig         : std_logic;

  component Rex210 port
    (
      Mclk     : in  std_logic;  -- Master clock 20MHz
      Reset    : in  std_logic;  -- Master Reset
      Xaxis    : in  std_logic;  -- X axis data
      Yaxis    : in  std_logic;  -- Y axis data
      Xdata    : out std_logic_vector(9 downto 0);
      Ydata    : out std_logic_vector(9 downto 0);
      Xgate    : out std_logic;  -- X sample gate
      Ygate    : out std_logic   -- Y sample gate
    );
  end component;

  component Tcsq port
    (
      Mclk     : in  std_logic;  -- Master clock 20MHz
      Reset    : in  std_logic;  -- Master Reset
      Xsend    : in  std_logic;  -- X data trigger
      Ysend    : in  std_logic;  -- Y data trigger
      Xdata    : in  std_logic_vector(9 downto 0);
      Ydata    : in  std_logic_vector(9 downto 0);
      Tgate    : out std_logic;  --- 20MHz/65 Gate pulse
      TxData   : out std_logic
    );
  end component;

  component Rcsq port
    (
      Mclk    : in  std_logic;  -- Master clock 20MHz
      Reset   : in  std_logic;  -- Master Reset
      Tgate   : in  std_logic;  -- 20MHz/65 gate pulse
      Rxd     : in  std_logic;  -- Serial input
      Xtrig   : out std_logic;  -- Trigger to send X data
      Ytrig   : out std_logic;  -- Trigger to send Y data
      Mtrig   : out std_logic;  -- Mode trigger
      Cal     : out std_logic;  -- Cal data
      Mode    : out std_logic_vector(1 downto 0)
    );
  end component;

  component Caldata port
    (
      Mclk    : in  std_logic;  -- Master clock 20MHz
      Reset   : in  std_logic;  -- Master Reset
      Xind    : in  std_logic_vector(9 downto 0);
      Yind    : in  std_logic_vector(9 downto 0);
      Xsamp   : in  std_logic;  -- X data Sample phase
      Ysamp   : in  std_logic;  -- Y data Sample phase
      Cal     : in  std_logic;  -- Cal data
      Mtrig   : in  std_logic;  -- Maximum
      Mode    : in  std_logic_vector(1 downto 0);
                                          -- Sample data
      Xout    : out std_logic_vector(9 downto 0);
      Yout    : out std_logic_vector(9 downto 0)
    );
  end component;

BEGIN
--------------------------------------------------------
--      Detect signal
--------------------------------------------------------
U0: Rex210 port map
    (
      Mclk     => Mclk,     -- Master clock 20MHz
      Reset    => Reset,    -- Master Reset
      Xaxis    => Xin,      -- X axis data
      Yaxis    => Yin,      -- Y axis data
      Xdata    => Xdata,
      Ydata    => Ydata,
      Xgate    => Xsamp,    -- X sample gate
```

リセットを割り当てます．そこで，リセットのピンにWeakPullUpを指定しました．このピン定義エディタは，VHDLで記述できない，電流駆動能力やその他のIC固有の属性を細かく設定できます．出力ピンは最低の4mAのドライブに設定しました．

FPGAの電源投入後，コンフィグレーションが終わるまで，全I/Oピンはフローティングになります．従って電源投入後は"L"で，コンフィグレーションが終わった後は，プルアップの抵抗がつながり，リセットが解除されます．さらにピンにコンデンサを付ければ，確実にリセットになりますが，シュミレーション時を除くとリセットはあまり重要ではありませんので，そこまでは行っていません．

```
        Ygate    => Ysamp    -- Y sample gate
      );

    --------------------------------------------------------
    --      Calculate data
    --------------------------------------------------------
    U1: Caldata port map
      (
        Mclk     => Mclk,     -- Master clock 20MHz
        Reset    => Reset,    -- Master Reset
        Xind     => Xdata,
        Yind     => Ydata,
        Xsamp    => Xsamp,    -- X data Sample phase
        Ysamp    => Ysamp,    -- Y data Sample phase
        Cal      => Cal,      -- Cal data
        Mtrig    => Mtrig,    -- Status trigger
        Mode     => Mode,     -- Process status
        Xout     => Xout,
        Yout     => Yout
      );

    --------------------------------------------------------
    --      Receive Serial Command
    --------------------------------------------------------
    U2: Rcsq port map
      (
        Mclk     => Mclk,     -- Master clock 12.288MHz
        Reset    => Reset,    -- Master Reset
        Tgate    => Tgate,    -- 20MHz/65 gate pulse
        Rxd      => Rxd,      -- Serial input
        Xtrig    => Xtrig,    -- Trigger to send X data
        Ytrig    => Ytrig,    -- Trigger to send Y data
        Mtrig    => Mtrig,    -- Mode trigger
        Cal      => Cal,      -- Cal data
        Mode     => Mode      -- Openration mode
      );

    --------------------------------------------------------
    --      Send Serial Data
    --------------------------------------------------------
    U3: Tcsq port map
      (
        Mclk     => Mclk,     -- Master clock 20MHz
        Reset    => Reset,    -- Master Reset
        Xsend    => Xtrig,    -- X data trigger
        Ysend    => Ytrig,    -- Y data trigger
        Xdata    => Xout,
        Ydata    => Yout,
        Tgate    => Tgate,    --- 20MHz/65 Gate pulse
        TxData   => Txd
      );
    END one;
```

● パルス幅読み取り部Rex210

パルス幅検出回路の処理ブロックを**図8**に示します．マスタ・クロックは20MHzですが，出力は1kHz，10ビット程度の解像度に設定するので，1/20のカウンタで1MHzのゲート信号を作ります．すべての回路はこの1MHzのゲート信号に同期して動作します．

次にADXL213から入力されるパルスの立ち上がりと立ち下がりを，**図9**で示すステート・マシンで検出します．ADXL213のパルスは，20MHzに対して非同期ですが，ここで20MHzに同期した，パルス・エッジ・タイミングが得られます．今回はパルスのチャタリング処理は入れていません．*X*軸（Xedge）と*Y*軸（Yedge）それぞれの回路があります．

パルスの立ち上がりエッジでリセットする1MHzのカウンタを使い，パルス幅を計測します．*X*軸（Xcount）と*Y*軸（Ycount）のそれぞれがあります．**図8**に示すように，このカウンタの値をパルスの立ち下がりでキャプチャします．取り込まれた値はそれぞれ*X*軸（Xwidth）と*Y*軸（Ywidth）にラッチされます．

● データ処理部Caldata

データ処理回路のブロックを**図10**に示します．Rex210ブロックから入力されるカウンタの値は，オフセット・バイナリですから，まずはそれを2の補数表現に変換します．処理は簡単で，0x200を加えればよいので，10ビットのMSB（most significant bit）の極性を反転するだけです．またRex210部からの出力データは，パルスの立ち下がりタイミングでデータが変化し都合が悪いので，常に同じタイミングになるように，変化しないパルスの立ち上がりでラッチします．

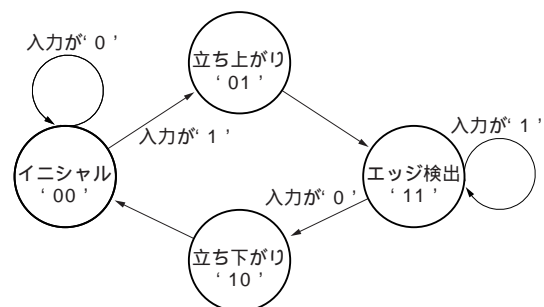
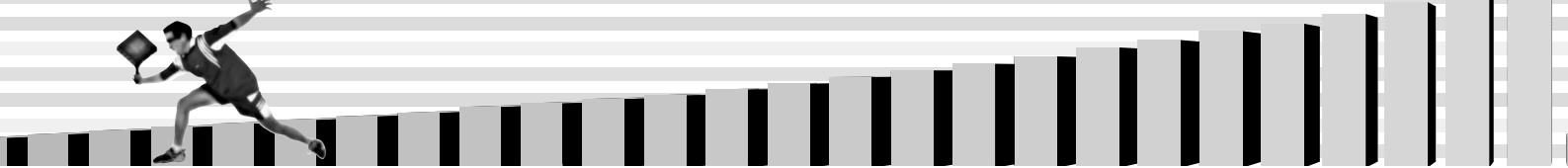



図9　立ち上がり/立ち下がり検出用ステート・ダイヤグラム
パルスのチャタリング処理は入れてない．

次にキャリブレーションのオフセット値を取り込む処理です．パソコンからのコマンドを受けたタイミングで，その時点の10ビットの値をX（Calx）/Y（Caly）にそれぞれ記憶します．データをキャリブレーションするため，現在の値から記憶したオフセット値（Calx，Caly）をそれぞれ引きます．これが現在値として出力されます．

絶対値の最大値を検出します．パソコンから最大値ホールド・コマンドがくると，その時点から変化した値のうち，絶対値の最大値を保持します．処理的にはこれまでの最大値から現在の値の引き算を行い，結果が負になった場合，現在値を最大値とします．

演算部の処理は，パソコンからコマンドを受け取ったとき，出力モードが変化します．モードはModeレジスタの中に保存されています．

00：現在値（キャリブレーションを取った後）を出力
01：現在までの最大値を出力
10：コマンドがきた時点のデータを保存し，出力

このModeやCalx，Calyはパソコンからのクリア・コマンドによってゼロに初期化されます．

● データ受信およびコマンド解釈部Rcsq

Rcsqモジュールの下には，非同期シリアルを受信するuartrxモジュールがあります．19.2kbpsで入力される「8ビット非同期，1ストップ・ビット，パリティなし」のシリアル・データを受信します．19.2kbpsのデータはその約16倍の307.692kHz（20MHz/65）でサンプリングして同期を検出し，受信されます．このようなUARTの回路構成はごく標準的なもので，皆さんも一度は作ったことがあるのではないでしょうか．

非同期シリアル受信部で受け取った8ビットの1文字データを，パソコンからのコマンドとして，どのコマンドに対応するかが判断されます．6種類のコマンドに対応するASCII文字を，CASE文を使って判断します．大文字だけでなく小文字でも入力できるように，合計12通りを判断できる回路です．

● 数値変換およびデータ送信部Tscq

処理のブロックを**図7**に示しました．このモジュールの下にも非同期シリアル・データを送信するuarttxモジュールがあります．データの送信トリガSloadが入力されると，19.2kbpsの非同期シリアル・データとして出力されます．同じく20［MHz］÷65÷16＝19.23［kHz］のクロックで1ビットずつ送り出されます．

ADXL213のデータは符号を含み10ビットで取り込みます．まずはそれを符号および3けたの10進数に変換します．これら一連の処理は**図11**で示すステート・ダイヤグラムで処理されます．ステート・マシンは4ビットのHstaで現在の状態が示されます．まずは絶対値に変換された後，100のけたから計算されます．100を何回引けるかで，100のけたが決まります．次に10のけたです．先ほどの残りの100以下の数値から，同じく10を何回引けるかで，10のけたが決まります．残りが1のけたとなります．

その後，符号（プラス/マイナス）および3けたの変換されたASCII数字は，データの変換速度とシリアル送信のタイミングを切り離すため，とりあえずFIFOの中に順次書き込まれます．FIFOの深さは16バイトと少ないですが，

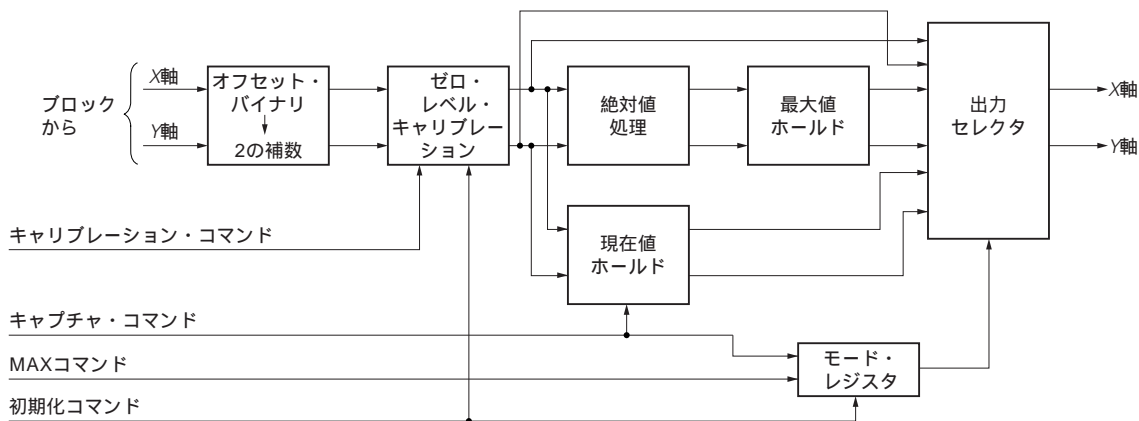



図10　データ処理回路のブロック図
Rex210ブロックから入力された値を2の補数に変換．その後，絶対値処理や最大値ホールドを行う．

これは人間の手入力程度では十分の深さです．FIFOの出力側では，FIFOが空になるまで，FIFOから1バイトずつ読み出され，uarttxモジュールからシリアル・データとして送信されます．

5．パソコンに接続して動作を確認

シリアルROMに回路データを書いた後，実際にパソコンにつないで動作確認をしてみます．**写真4**のように一般的な9ピンのシリアル・ケーブルを使い，パソコンのCOMに接続します．筆者の場合は，ターミナル・ソフトとして，フリーのソフトウェア「WTERM」を使っています．立ち上げ通信条件を，「通信速度：19.2kBaud，データ：8ビット，パリティなし，1ストップ・ビット」に設定します．

とりあえず，実験基板を地面に対して水平に置いた状態で，5Vの電源を投入します．そこでX軸のデータを読みとるため，*X*のキーを押してみます．すると，**図12**のように，現在の測定値がターミナルの画面に表示されます．

本当は水平に置いているため重力加速度の影響はないはずです．しかし，ゼロにはなりません．理由は，ADXL213のパルス幅が正確に1kHzになっていないことと，ICのばらつきでしょう．そこで，この状態を加速度ゼロとするため，キャリブレーション・コマンドとして［C］キーを押してみます．その後同じく*X*軸の値を読み取るため［X］キーを押すと，**図13**のように，今度はほぼゼロの値を示すようになります．次に*Y*軸の値を読むため，［Y］キーを押してみます．同じくキャリブレーションされたゼロ近くの値が表示されるはずです．

今度は重力加速度を測って見ます．加速度センサの*X*軸正方向に基板を垂直に立てて（**写真5**），*X*軸の値を読み取ります．すると，**図14**のように，今度はゼロではなく重力加速度1*g*に対応する値が返されてきます．**図14(a)**のはじめの二つの値は，**写真5**の位置で読み取った*X*と*Y*の値です．次の二つの値は，基板を**写真5**とはさかさまにして計ったときの*X*と*Y*の値です．ここで*Y*軸の値を読み取ってみると，ゼロに近い値を示します．加速度センサの*Y*軸方向は水平だからです．

次に加速度センサの*X*軸方向は水平に，*Y*軸方向が垂直になるように基板を立てて見ます．その状態で*Y*軸の読み取りコマンドを送ると，**図14(b)**のような値が読み取れるはずです．*X*軸はゼロ付近を示します．ADXL213の加速度センサが*X*軸と*Y*軸とで，直交して2個配置されていることがよくわかります．

＊　＊　＊

今回はADXL213を実際に動作させてみて，その働きを体験してみました．思いのほか簡単に使えたというのが率直な感想です．特にADXL213の場合，出力がPWMになっているため，マイコンやロジックICと直接接続するこ

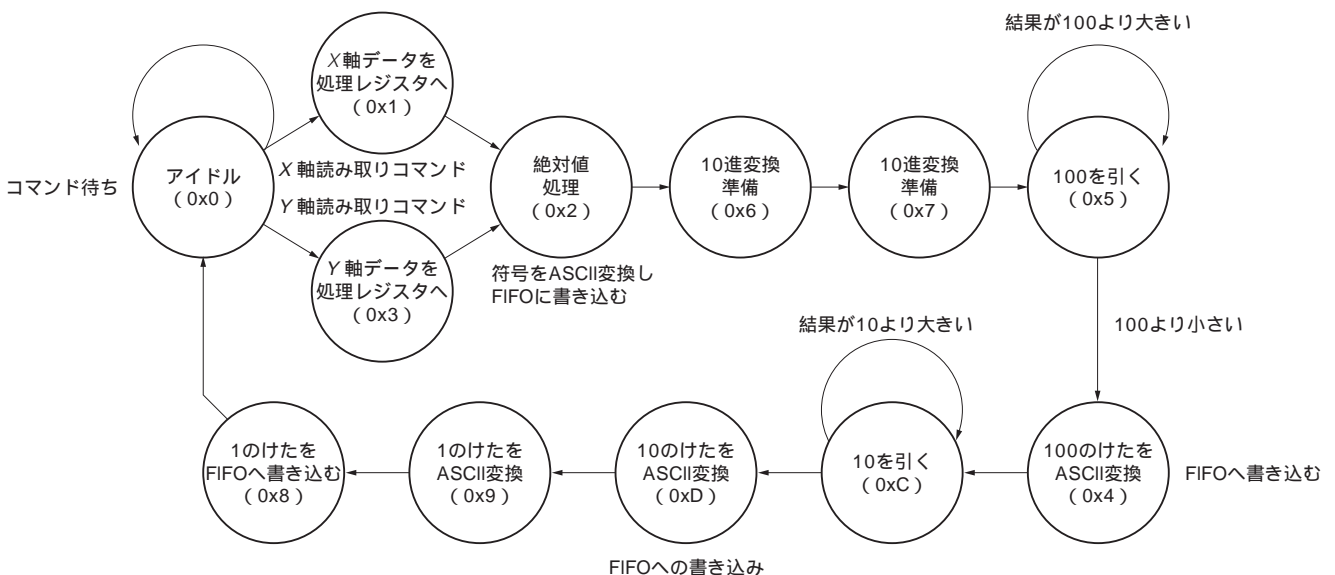



図11　*X*/*Y*データの10進変換とFIFO書き込みシーケンス
ADXL213のデータは符号を含み10ビットで取り込む．まずはそれを符号および3けたの10進数に変換する．

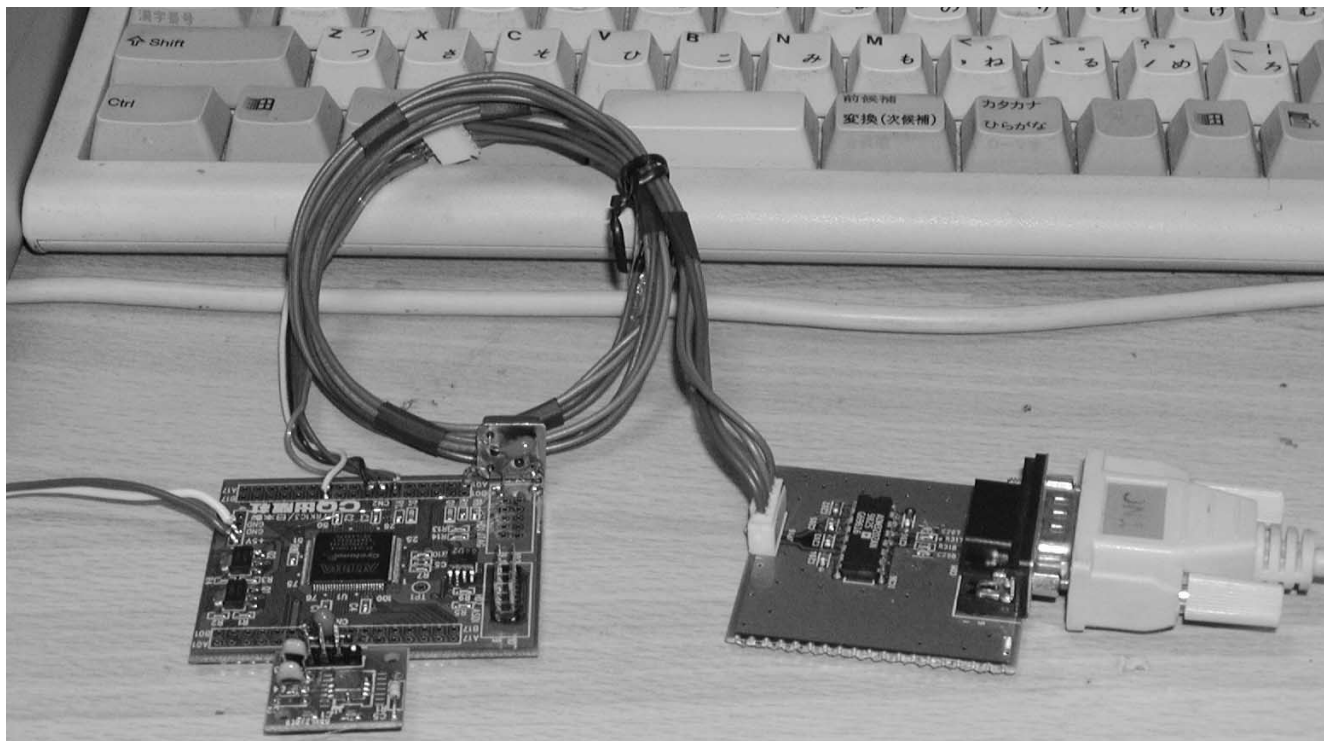



写真4　2003年10月号付属基板とレベル変換基板をパソコンに接続

図12　電源投入後の測定値

ゼロにはならない．理由はADXL213のパルス幅が正確に1kHzになっていないことと，センサのばらつきがあるため．

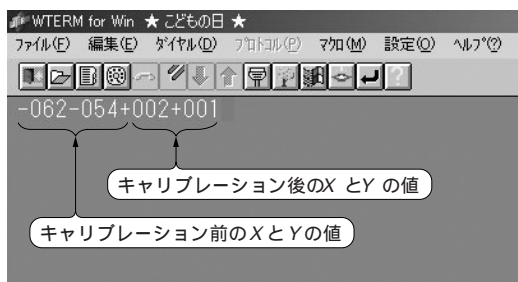



図13　キャリブレーション後の値

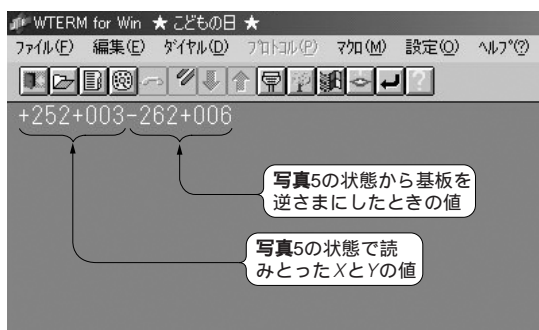
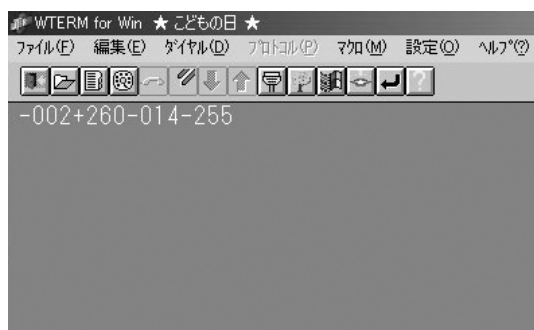
はじめの二つの数値はキャリブレーション前のX，Yの値．次の二つの数値はキャリブレーション後のX，Yの値である．

（a）X軸を垂直に立てたときの値

（b）Y軸を垂直に立てたときの値

図14　重力加速度に対応する値

とができ，とても便利だと感じました．

加速度センサはアナログ・レベル出力のものが多いようです．多くのマイコンには，通常，A-Dコンバータは付いていますから，そんなに大きな問題とはならないと思いますが，やはりロジック・レベルで接続できるのは便利です．

実験基板を使っていて，いろいろな応用が頭に浮かんできました．例えば，平面の傾きセンサなどが簡単に製作できそうです．皆さんもこれを機会に面白い製品を設計できるのではないでしょうか．

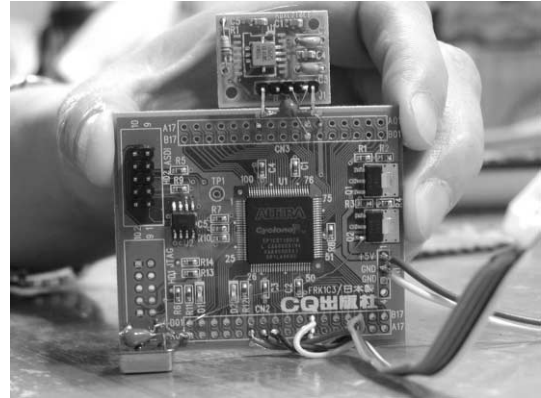



写真5　重力加速度を測定するため基板を垂直に立てたようす

にしむら・よしかず
(株)エー・オー・アール

参考・引用*文献

(1)* ADXL213データシート，Analog Devices.
http://www.analog.com/UploadedFiles/Data_Sheets/ADXL213.pdf
(2)* ADXL213EBデータシート，Analog Devices.
http://www.analog.com/UploadedFiles/Evaluation_Boards_Tools/53621793629198ADXL213EB_0.pdf

<筆者プロフィール>
西村芳一．1954年，長崎県長崎市生まれ．現在，エーオーアールにて，受信機の開発に従事．最近のディジタル化と信号の高周波化で，ますます勉強しないといけないことが山積み．関連書籍はとりあえずそろえたが，元来のグータラ症で，「まーいいか，明日にしよう！…」．脳の空洞化現象が進むいっぽう．
ja6uhl@cqsstv.com

3